* **"STORAGE"** thường liên quan đến việc quản lý và lưu trữ dữ liệu trong ứng dụng của bạn. Dưới đây là một số khái niệm cơ bản về lưu trữ trong Flutter:

1. SharedPreferences: Đây là một cách để lưu trữ dữ liệu nhỏ dưới dạng cặp khóa-giá trị. SharedPreferences thường được sử dụng để lưu trữ các cài đặt người dùng hoặc thông tin phiên làm việc như mã thông báo đăng nhập.
2. File System: Bạn có thể sử dụng hệ thống tệp để lưu trữ dữ liệu trong các tệp tin. Flutter cung cấp một API để ghi và đọc tệp tin, cho phép bạn lưu trữ dữ liệu dưới dạng văn bản hoặc nhị phân.
3. Local Database: Để lưu trữ dữ liệu phức tạp hơn, như danh bạ người dùng hoặc các mục trong ứng dụng, bạn có thể sử dụng cơ sở dữ liệu local. Flutter hỗ trợ các thư viện như sqflite hoặc moor để làm việc với cơ sở dữ liệu SQLite.
4. External Storage: Đối với các ứng dụng có liên quan đến các tệp tin hoặc phương tiện lưu trữ lớn, bạn có thể sử dụng bộ nhớ ngoại vi như thẻ SD để lưu trữ dữ liệu.
5. Đám mây và Dịch vụ Lưu trữ: Bạn cũng có thể tích hợp các dịch vụ lưu trữ đám mây như Firebase để lưu trữ và quản lý dữ liệu của ứng dụng.

**Ví dụ về cách sử dụng sharedpreferences để lưu trữ cài đặt người dùng:**

*import 'package:flutter/material.dart';*

*import 'package:shared\_preferences/shared\_preferences.dart';*

*void main() {*

*runApp(MyApp());*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*title: 'Flutter Storage Demo',*

*home: StorageDemo(),*

*);*

*}*

*}*

*class StorageDemo extends StatefulWidget {*

*@override*

*\_StorageDemoState createState() => \_StorageDemoState();*

*}*

*class \_StorageDemoState extends State<StorageDemo> {*

*late SharedPreferences \_prefs;*

*String \_username = '';*

*@override*

*void initState() {*

*super.initState();*

*\_initPrefs();*

*}*

*Future<void> \_initPrefs() async {*

*\_prefs = await SharedPreferences.getInstance();*

*setState(() {*

*\_username = \_prefs.getString('username') ?? '';*

*});*

*}*

*void \_updateUsername(String newUsername) {*

*\_prefs.setString('username', newUsername);*

*setState(() {*

*\_username = newUsername;*

*});*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Scaffold(*

*appBar: AppBar(*

*title: Text('Storage Demo'),*

*),*

*body: Center(*

*child: Column(*

*mainAxisAlignment: MainAxisAlignment.center,*

*children: [*

*Text('Username: $\_username'),*

*SizedBox(height: 20),*

*ElevatedButton(*

*onPressed: () {*

*\_updateUsername('new\_user');*

*},*

*child: Text('Change Username'),*

*),*

*],*

*),*

*),*

*);*

*}*

*}*

chúng ta sử dụng **SharedPreferences** để lưu trữ và cập nhật tên người dùng. Khi người dùng bấm vào nút **"Change Username**", tên người dùng sẽ được cập nhật và lưu lại trong **SharedPreferences**.

* **"3RD PARTY LIBRARIES"** (thư viện bên thứ ba) là các gói mã nguồn mở được phát triển bởi các nhà phát triển khác nhau, không phải là người phát triển chính của Flutter, nhưng có thể được tích hợp vào dự án của bạn để cung cấp các tính năng, chức năng và giao diện người dùng bổ sung. Điều này giúp bạn tiết kiệm thời gian và công sức trong việc xây dựng các phần của ứng dụng.

Dưới đây là một số khái niệm cơ bản liên quan đến 3rd Party Libraries trong Flutter:

1. Pub.dev: Đây là trang web chính thức để tìm kiếm, khám phá và tải xuống các thư viện Flutter bên thứ ba. Trang web này cung cấp danh sách các gói thư viện khác nhau, bao gồm cả mã nguồn mở và thương mại.
2. Dependency Management: Khi bạn sử dụng một thư viện bên thứ ba trong dự án Flutter của mình, bạn cần khai báo thư viện đó là phụ thuộc (dependency) của dự án. Điều này thường được thực hiện trong tệp pubspec.yaml bằng cách thêm tên và phiên bản của thư viện vào danh sách dependencies.
3. Installation and Import: Sau khi đã khai báo phụ thuộc, bạn có thể chạy lệnh flutter pub get để tải về và cài đặt các thư viện mới. Sau đó, bạn có thể nhập (import) các thành phần từ thư viện vào mã nguồn của mình để sử dụng.
4. Common Use Cases: Các thư viện bên thứ ba trong Flutter có thể đáp ứng nhiều nhu cầu khác nhau, chẳng hạn như:

Xây dựng giao diện người dùng phức tạp hơn với các widget tùy chỉnh.

Xử lý các tác vụ như HTTP requests, gửi email, làm việc với cơ sở dữ liệu, quản lý trạng thái ứng dụng, và nhiều tính năng khác.

Điều hướng và quản lý đường dẫn (routing) giữa các màn hình.

Xử lý hình ảnh, âm thanh, video và phương tiện truyền thông khác.

Tích hợp các dịch vụ như Firebase, GraphQL, OAuth, và nhiều dịch vụ khác.

1. Thư viện phổ biến: Một số thư viện phổ biến trong cộng đồng Flutter bao gồm:

http: Để thực hiện các yêu cầu HTTP.

provider: Để quản lý trạng thái ứng dụng và cung cấp dữ liệu đến các widget con.

shared\_preferences: Để lưu trữ dữ liệu người dùng.

firebase\_core và các gói Firebase khác: Để tích hợp với các dịch vụ của Firebase.

flutter\_bloc và mobx: Để quản lý trạng thái ứng dụng với kiến trúc Bloc hoặc MobX.

* **"BEHAVIOR COMPONENTS"** (Các thành phần Hành vi) thường liên quan đến cách mà các phần của giao diện người dùng hoạt động và tương tác với người dùng. Đây là một số thành phần hành vi quan trọng trong Flutter:

1. Gesture Detector (Bộ dò cử chỉ): GestureDetector cho phép bạn bắt các cử chỉ tương tác như chạm, vuốt, nhấn giữ và nhiều loại cử chỉ khác trên các widget. Bạn có thể sử dụng GestureDetector để thêm khả năng tương tác vào các widget.
2. InkWell (Bề mặt mực): InkWell là một widget thường được sử dụng để tạo ra hiệu ứng hồi ứng (ripple effect) khi người dùng chạm vào nó. Điều này tạo ra trải nghiệm tương tác thú vị và hợp thời với giao diện người dùng.
3. Draggable và DraggableScrollableSheet: Các widget Draggable cho phép người dùng kéo và thả một widget từ vị trí này sang vị trí khác. DraggableScrollableSheet cho phép bạn tạo một widget có thể kéo lên và xuống trên màn hình, giống như thanh cuộn.
4. Scrolling Behavior (Hành vi cuộn): Các widget có hành vi cuộn như ListView, GridView, SingleChildScrollView, và CustomScrollView cho phép bạn hiển thị nội dung lớn hơn màn hình và cho phép người dùng cuộn qua nội dung đó.
5. Animations (Hoạt ảnh): Flutter cung cấp một hệ thống mạnh mẽ để tạo các hoạt ảnh tương tác và chuyển động. Bạn có thể sử dụng các thư viện như flutter\_animation\_progressions hoặc tạo hoạt ảnh tùy chỉnh bằng cách sử dụng các API Animation có sẵn.
6. Listener Widgets (Các widget lắng nghe sự kiện): Các widget như Listener cho phép bạn lắng nghe các sự kiện tương tác như chạm, vuốt, di chuyển và nhiều sự kiện khác trên widget cha và các widget con.
7. Focus và FocusScope (Tiêu điểm và Phạm vi tiêu điểm): Focus và FocusScope được sử dụng để quản lý tiêu điểm (focus) của các widget, chẳng hạn như khi người dùng nhập liệu vào các trường văn bản hoặc điều hướng bằng bàn phím.
8. PopupMenuButton và DropdownButton: Các widget này cho phép bạn tạo các menu hoặc danh sách thả xuống (dropdown) để chọn một tùy chọn từ danh sách.
9. SnackBar và BottomSheet: SnackBar là một hình thức thông báo ngắn xuất hiện ở dưới cùng của màn hình, trong khi BottomSheet là một lớp hiển thị lên trên màn hình để hiển thị thông tin hoặc tương tác thêm.

* **QUẢN LÝ TRẠNG THÁI (STATE MANAGEMENT)** trong Flutter là một khía cạnh quan trọng khi xây dựng ứng dụng. Trạng thái đề cập đến dữ liệu và thông tin hiện tại của ứng dụng, bao gồm cả trạng thái giao diện người dùng và dữ liệu liên quan. Quản lý trạng thái đúng cách giúp bạn duy trì tính mở rộng, dễ bảo trì và tối ưu hiệu suất của ứng dụng.

Dưới đây là một số khái niệm cơ bản về quản lý trạng thái trong Flutter:

1. Local State (Trạng thái cục bộ): Trạng thái cục bộ là trạng thái nội tại của một widget cụ thể. Nó thường được quản lý bởi StatefulWidget. Khi trạng thái cục bộ thay đổi, widget sẽ được xây dựng lại để hiển thị trạng thái mới.
2. InheritedWidget và InheritedModel: Đây là các cơ chế mà bạn có thể sử dụng để chia sẻ trạng thái giữa các widget con một cách hiệu quả. InheritedWidget cho phép bạn chia sẻ trạng thái qua cây widget mà không cần truyền nó qua các hàm xây dựng.
3. ScopedModel: ScopedModel là một thư viện được thiết kế để giúp quản lý trạng thái của ứng dụng bằng cách sử dụng các lớp model. Nó giúp bạn truyền trạng thái xuống từ gốc cây widget đến các widget con.
4. Provider: Thư viện Provider là một cách phổ biến để quản lý trạng thái trong Flutter. Nó dựa trên InheritedWidget và cung cấp cách dễ dàng để cung cấp và truy cập trạng thái cho các widget con.
5. Redux: Redux là một kiến trúc quản lý trạng thái phức tạp hơn, thường được sử dụng cho các ứng dụng lớn hoặc có trạng thái phức tạp. Nó sử dụng các khái niệm như "store", "actions" và "reducers" để quản lý trạng thái và tương tác với nó.
6. BLoC (Business Logic Component): BLoC là một mô hình quản lý trạng thái được đề xuất bởi cộng đồng Flutter. Nó sử dụng các lớp "Bloc" để quản lý trạng thái và luồng dữ liệu trong ứng dụng.
7. GetX: GetX là một thư viện mạnh mẽ và dễ sử dụng để quản lý trạng thái và điều hướng trong Flutter. Nó cung cấp nhiều tính năng như quản lý trạng thái, điều hướng, dependency injection và nhiều chức năng khác.
8. Riverpod: Riverpod là một thư viện quản lý trạng thái mới với cú pháp dễ hiểu và hiệu suất cao. Nó tương tự như Provider nhưng cung cấp một cách tiếp cận khác để quản lý trạng thái.

* **QUALITY ASSURANCE (QA)** là quá trình kiểm tra và đảm bảo chất lượng của phần mềm, bao gồm cả ứng dụng Flutter. Trong ngữ cảnh của Flutter, Quality Assurance bao gồm các hoạt động để đảm bảo rằng ứng dụng của bạn hoạt động một cách đáng tin cậy, hiệu quả và tuân thủ các tiêu chuẩn chất lượng. Dưới đây là một số khái niệm cơ bản về Quality Assurance trong Flutter:

1. Unit Testing (Kiểm thử đơn vị): Là quá trình kiểm tra từng phần riêng lẻ của mã nguồn để đảm bảo rằng chúng hoạt động đúng theo thiết kế. Trong Flutter, bạn có thể sử dụng các framework như flutter\_test để viết và chạy các bài kiểm thử đơn vị.
2. Integration Testing (Kiểm thử tích hợp): Kiểm tra sự tương tác giữa các thành phần khác nhau của ứng dụng để đảm bảo tích hợp hoạt động một cách chính xác. flutter\_driver là một phần của Flutter cho phép bạn thực hiện kiểm thử tích hợp.
3. Widget Testing (Kiểm thử widget): Đối với Flutter, widget testing kiểm tra hành vi của các widget và giao diện người dùng trong ứng dụng. Bạn có thể sử dụng flutter\_test để viết các kiểm thử widget.
4. Automated Testing (Kiểm thử tự động): Sử dụng mã nguồn để thực hiện các kiểm thử một cách tự động, giúp giảm thời gian và công sức so với kiểm thử thủ công. Các loại kiểm thử như kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử widget đều có thể được tự động hóa.
5. Manual Testing (Kiểm thử thủ công): Kiểm thử được thực hiện bằng tay để kiểm tra các khía cạnh của ứng dụng mà không thể tự động hóa một cách hiệu quả. Điều này bao gồm việc thử nghiệm từng phần của ứng dụng, kiểm tra tích hợp, hiệu suất và giao diện người dùng.
6. Performance Testing (Kiểm thử hiệu suất): Đảm bảo rằng ứng dụng Flutter hoạt động một cách hiệu quả về mặt tốc độ và tài nguyên. Kiểm thử hiệu suất đảm bảo rằng ứng dụng không gặp vấn đề về tốc độ phản hồi, xử lý dữ liệu lớn và tài nguyên hệ thống.
7. Usability Testing (Kiểm thử tính sử dụng): Kiểm tra sự dễ sử dụng và trải nghiệm người dùng của ứng dụng. Điều này đảm bảo rằng ứng dụng có giao diện người dùng thân thiện và dễ dàng sử dụng.
8. Security Testing (Kiểm thử bảo mật): Kiểm tra tính bảo mật của ứng dụng, đảm bảo rằng dữ liệu và thông tin cá nhân của người dùng được bảo vệ.
9. Regression Testing (Kiểm thử hồi quy): Đảm bảo rằng các thay đổi trong mã nguồn hoặc cập nhật không gây ra lỗi mới trong các tính năng đã tồn tại.
10. Code Review (Xem xét mã nguồn): Đánh giá và kiểm tra mã nguồn của ứng dụng bằng cách sử dụng xem xét từ đồng nghiệp hoặc người khác trong nhóm phát triển.

* **VERSION CONTROL (QUẢN LÝ PHIÊN BẢN**) là một hệ thống để theo dõi, quản lý và kiểm soát phiên bản của mã nguồn trong quá trình phát triển phần mềm. Trong ngữ cảnh của Flutter, quản lý phiên bản là cách để theo dõi và lưu trữ mã nguồn của ứng dụng Flutter của bạn và theo dõi sự thay đổi và cải tiến qua thời gian. Một hệ thống quản lý phiên bản giúp bạn làm việc cùng nhóm, tạo nhánh (branch) để phát triển đồng thời, kiểm soát sự thay đổi, và đảm bảo tính ổn định của ứng dụng.

Dưới đây là một số khái niệm cơ bản về Version Control trong Flutter:

1. Repositories (Kho mã nguồn): Một kho mã nguồn là nơi bạn lưu trữ mã nguồn của ứng dụng và tất cả các thông tin liên quan đến quản lý phiên bản. Popular version control hosting services include GitHub, GitLab, and Bitbucket.
2. Commits (Lần ghi): Mỗi lần bạn thay đổi mã nguồn và muốn lưu trạng thái hiện tại của mã, bạn sẽ thực hiện một commit. Mỗi commit chứa một bản chụp của tất cả các tập tin đã thay đổi và một tin nhắn mô tả về thay đổi.
3. Branches (Nhánh): Branches cho phép bạn phát triển và thử nghiệm mã nguồn mà không ảnh hưởng đến phiên bản chính của ứng dụng. Bạn có thể tạo và chuyển đổi giữa các nhánh để phát triển đồng thời và thử nghiệm tính năng mới.
4. Merging (Hợp nhất): Khi bạn muốn kết hợp các thay đổi từ một nhánh vào nhánh khác, bạn thực hiện một hợp nhất (merge). Hợp nhất đảm bảo rằng các thay đổi được tích hợp một cách đúng đắn.
5. Pull Requests/Merge Requests (Yêu cầu hợp nhất): Một quá trình gửi yêu cầu để hợp nhất các thay đổi từ một nhánh vào nhánh khác. Người khác trong nhóm có thể xem xét và phê duyệt yêu cầu hợp nhất trước khi thay đổi thực sự được hợp nhất.
6. Version Tags (Nhãn phiên bản): Một phiên bản được gán một nhãn (tag) để đánh dấu một trạng thái cụ thể của ứng dụng. Điều này giúp bạn dễ dàng theo dõi và xác định phiên bản trong lịch sử phát triển.
7. Conflict Resolution (Giải quyết xung đột): Khi hai người cùng chỉnh sửa cùng một phần của mã nguồn, xung đột có thể xảy ra khi hợp nhất. Giải quyết xung đột là quá trình sửa các xung đột này một cách thích hợp.
8. Version Control Systems (Hệ thống quản lý phiên bản): Trong Flutter, Git là hệ thống quản lý phiên bản phổ biến nhất. Flutter sử dụng Git để theo dõi và quản lý mã nguồn.

* **FIREBASE** là một nền tảng phát triển ứng dụng đám mây do Google cung cấp. Nó cung cấp một loạt các dịch vụ và công cụ để phát triển ứng dụng di động và web một cách dễ dàng và nhanh chóng. Trong ngữ cảnh của Flutter, Firebase là một giải pháp phổ biến để tích hợp các tính năng và dịch vụ như cơ sở dữ liệu thời gian thực, xác thực người dùng, lưu trữ tệp, phân tích và nhiều chức năng khác vào ứng dụng của bạn.

Dưới đây là một số khái niệm cơ bản về Firebase trong Flutter:

1. Firebase Authentication (Xác thực Firebase): Cho phép bạn thêm tính năng xác thực người dùng bằng cách sử dụng các phương thức như đăng nhập bằng email, số điện thoại, Google, Facebook, Twitter, và nhiều tùy chọn khác.
2. Cloud Firestore và Realtime Database: Đây là hai loại cơ sở dữ liệu thời gian thực trong Firebase. Bạn có thể lưu trữ và đồng bộ hóa dữ liệu của ứng dụng ngay lập tức trên các thiết bị khác nhau.
3. Firebase Storage: Dịch vụ lưu trữ tệp cho phép bạn lưu trữ các tệp như hình ảnh, video và tài liệu trên đám mây Firebase.
4. Firebase Cloud Functions: Cho phép bạn viết mã nguồn để xử lý sự kiện và logic trên máy chủ mà không cần quản lý cơ sở hạ tầng máy chủ riêng.
5. Firebase Cloud Messaging (FCM): Dịch vụ thông báo đám mây cho phép bạn gửi thông báo đến các thiết bị di động và web.
6. Firebase Analytics: Cung cấp phân tích ứng dụng để bạn theo dõi và đo lường hiệu suất ứng dụng của mình.
7. Firebase Performance Monitoring: Giúp bạn theo dõi và cải thiện hiệu suất ứng dụng bằng cách theo dõi thời gian phản hồi, thời gian tải và thời gian xử lý.
8. Firebase Remote Config: Cho phép bạn thay đổi cài đặt và nội dung ứng dụng từ xa mà không cần phải phát hành phiên bản mới.
9. Firebase Hosting: Cung cấp dịch vụ lưu trữ và phân phối ứng dụng web của bạn.

Để tích hợp Firebase vào ứng dụng Flutter, cần thêm các gói firebase vào tệp **pubspec.yaml** của dự án và cấu hình Firebase trong dự án của mình. Firebase cung cấp tài liệu chi tiết và hướng dẫn để giúp bạn bắt đầu tích hợp và sử dụng các dịch vụ của nó trong ứng dụng Flutter.

* **NATIVE INTEGRATION** trong Flutter đề cập đến việc tích hợp mã nguồn native (Java/Kotlin cho Android, Swift/Objective-C cho iOS) vào ứng dụng Flutter của bạn để sử dụng các tính năng hoặc thư viện không có sẵn trong Flutter Framework. Điều này cho phép bạn tận dụng các tính năng mạnh mẽ của các nền tảng native mà không cần phải tạo lại hoặc sử dụng thư viện bên thứ ba.

Dưới đây là một số khía cạnh cơ bản về Native Integration trong Flutter:

1. Platform Channels (Kênh nền tảng): Flutter cung cấp "platform channels" để trao đổi dữ liệu và gọi phương thức giữa mã Dart trong Flutter và mã native. Bạn có thể sử dụng platform channels để gọi mã native từ Flutter hoặc trả về kết quả từ mã native.
2. Method Channels (Kênh phương thức): Kênh phương thức là một phần của platform channel, cho phép bạn gọi các phương thức native từ mã Dart và nhận kết quả trả về. Điều này thường được sử dụng để gọi các tính năng native như máy ảnh, máy quét mã vạch, cảm biến thiết bị, và nhiều chức năng khác.
3. Event Channels (Kênh sự kiện): Kênh sự kiện cũng là một phần của platform channel, cho phép bạn lắng nghe và truyền sự kiện từ mã native đến mã Dart. Điều này thường được sử dụng khi bạn cần lắng nghe các sự kiện như cảm biến, kết nối mạng, hoặc bất kỳ sự kiện native nào khác.
4. Native Modules/Plugins: Bạn có thể tạo các module native hoặc plugins để tích hợp các tính năng hoặc thư viện native vào ứng dụng Flutter. Điều này giúp bạn tái sử dụng mã nguồn native và quản lý tích hợp một cách tốt hơn.
5. Platform Views (Giao diện nền tảng): Flutter cung cấp khả năng hiển thị giao diện native (như một khung nhìn Android hoặc một view iOS) bên trong ứng dụng Flutter bằng cách sử dụng PlatformView.
6. Using Existing Native Codebase (Sử dụng mã nguồn native hiện có): Bạn có thể tích hợp ứng dụng Flutter vào mã nguồn native hiện có hoặc ngược lại bằng cách sử dụng tích hợp như Flutter Module hoặc Flutter Package.